Learning Maps for Indoor Mobile Robot Navigation

Sebastian Thrun Arno Bücken

April 1996

CMU-CS-96-121

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

The first author is partly and the second author exclusively affiliated with the Computer Science Department III of the University of Bonn, Germany, where part of this research was carried out.

This research is sponsored in part by the National Science Foundation under award IRI-9313367, and by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of NSF, Wright Laboratory or the United States Government.

Keywords: autonomous robots, exploration, mobile robots, neural networks, occupancy grids, path planning, planning, robot mapping, topological maps

Abstract

Autonomous robots must be able to learn and maintain models of their environments. Research on mobile robot navigation has produced two major paradigms for mapping indoor environments: grid-based and topological. While grid-based methods produce accurate metric maps, their complexity often prohibits efficient planning and problem solving in large-scale indoor environments. Topological maps, on the other hand, can be used much more efficiently, yet accurate and consistent topological maps are considerably difficult to learn in large-scale environments.

This paper describes an approach that integrates both paradigms: grid-based and topological. Grid-based maps are learned using artificial neural networks and Bayesian integration. Topological maps are generated on top of the grid-based maps, by partitioning the latter into coherent regions. By combining both paradigms—grid-based and topological—, the approach presented here gains the best of both worlds: accuracy/consistency and efficiency. The paper gives results for autonomously operating a mobile robot equipped with sonar sensors in populated multi-room environments.

1 Introduction

To efficiently carry out complex missions in indoor environments, autonomous mobile robots must be able to acquire and maintain models of their environments. The task of acquiring models is difficult and far from being solved. The following factors impose practical limitations on a robot's ability to learn and use accurate models:

1. **Sensors.** Sensors often are not capable of directly measuring the quantity of interest. For example, cameras measure color, brightness and saturation of light, whereas for navigation, one might be interested in assertions such as *"there is a door in front of the robot."*

2. **Perceptual limitations.** The perceptual range of most sensors (such as ultrasonic transducers, cameras) is limited to a small range around the robot. To acquire global information, the robot has to actively explore its environment.

3. **Sensor noise.** Sensor measurements are typically corrupted by noise. Often, the distribution of this noise is unknown (it is rarely Gaussian).

4. **Drift/slippage.** Robot motion is inaccurate. Unfortunately, odometric errors accumulate with time. For example, even the smallest rotational errors can have huge effects on subsequent translational error when estimating the robot's position.

5. **Complexity and dynamics.** Robot environments are complex and dynamic, making it principally impossible to maintain exact models.

6. **Real-time requirements.** Time requirements often demand that the internal model must be simple and easily accessible. For example, accurate fine-grain CAD models of complex indoor environments are often disadvantageous if actions have to be generated quickly.

Recent research has produced two fundamental paradigms for modeling indoor robot environments: the *grid-based (metric) paradigm* and the *topological paradigm*. Grid-based approaches, such as those proposed by Moravec/Elfes [19] and Borenstein/Koren [2] and many others, represent environments by evenly-spaced grids. Each grid cell may, for example, indicate the presence of an obstacle in the corresponding region of the environment. Topological approaches, such a those described in [7, 14, 15, 17, 22, 35], represent robot environments by graphs. Nodes in such graphs correspond to distinct situations, places, or landmarks (such as doorways). They are connected by arcs if there exists a direct path between them.

Both approaches to robot mapping exhibit orthogonal strengths and weaknesses. Occupancy grids are considerably easy to construct and to maintain even in large-scale environments [3]. Since the intrinsic geometry of a grid corresponds directly to the geometry of the environment, the robot's position within its model can be determined by its position and orientation in the real world—which, as shown below, can be determined sufficiently accurately using only sonar sensors, in environments of moderate size. As a pleasing consequence, different positions for which sensors measure the same values (*i.e.*, situations that look alike) are naturally disambiguated in grid-based approaches. This is not the case for topological approaches, which determine the position of the robot relative to the model based on landmarks or distinct sensory features. For example, if the robot traverses two places that look alike, topological approaches often have difficulty determining if these places are the same or not (particularly if these places have been reached via different paths). Also, since sensory input usually depends strongly on the viewpoint of the robot, topological approaches may fail to recognize geometrically nearby places.

On the other hand, grid-based approaches suffer from their enormous space and time complexity. This is because the resolution of a grid must be fine enough to capture every important detail of the world. The key advantage of topological representation is their compactness. The resolution of topological maps corresponds directly to the complexity of the environment. The compactness of topological representations gives them three key advantages over grid-based approaches: (a) they permit fast planning, (b) they facilitate interfacing to symbolic planners and problem-solvers, and (c) they provide more natural interfaces for human instructions (such as: *"go to room A"*). Since topological approaches usually do not require the exact determination of the geometric position of the robot, they often recover better from drift and slippage—phenomena that must constantly be monitored and compensated in grid-based approaches. To summarize, both paradigms have orthogonal strengths and weaknesses, which are summarized in Table 1.

This paper advocates to integrate both paradigms, to gain the best of both worlds. The approach presented here combines grid-based (metric) and topological representations. To construct a grid-based model of the environment, sensor values are interpreted by an artificial neural network and mapped into probabilities for occupancy. Multiple interpretations are integrated over time using Bayes' rule. On top of the grid representation, more compact topological maps are generated by splitting the metric map into coherent regions, separated through *critical lines*. Critical lines correspond to narrow passages such as doorways. By partitioning the metric map into a small number of regions, the number of topological entities is several orders of magnitude smaller than the number of cells in the grid represen-

Grid-based (metric) approaches	**Topological approaches**
+ easy to build, represent, and maintain + recognition of places (based on geometry) is non-ambiguous and view point-independent + facilitates computation of shortest paths	+ permits efficient planning, low space complexity (resolution depends on the complexity of the environment) + does not require accurate determination of the robot's position + convenient representation for symbolic planner/problem solver, natural language
– planning inefficient, space-consuming (resolution does not depend on the complexity of the environment) – requires accurate determination of the robot's position – poor interface for most symbolic problem solvers	– difficult to construct and maintain in larger environments – recognition of places (based on landmarks) often ambiguous, sensitive to the point of view – may yield suboptimal paths

Table 1: Advantages and disadvantages of grid-based and topological approaches to map building.

tation. Therefore, the integration of both representations has unique advantages that cannot be found for either approach in isolation: the grid-based representation, which is considerably easy to construct and maintain in environments of moderate complexity (e.g., 20 by 30 meters), models the world consistently and disambiguates different positions. The topological representation, which is grounded in the metric representation, facilitates fast planning and problem solving.

The robots used in our research are shown in Figure 1. Among other sensors, all robots are equipped with an array of 24 sonar sensors. Sonars sensors return the proximity of surrounding obstacles, along with noise. Throughout this paper, we will restrict ourselves to the interpretation of sonar sensors, although the methods described here have (in a prototype version) also been operated using cameras and infrared light sensors in addition to sonar sensors, using the image segmentation approach described in [3]. The integrated approach to map building with sonar sensors has extensively been tested in various indoor environments.

The remainder of the paper is organized as follows. Section 2 describes our approach for building grid-based maps, followed by the description of our approach

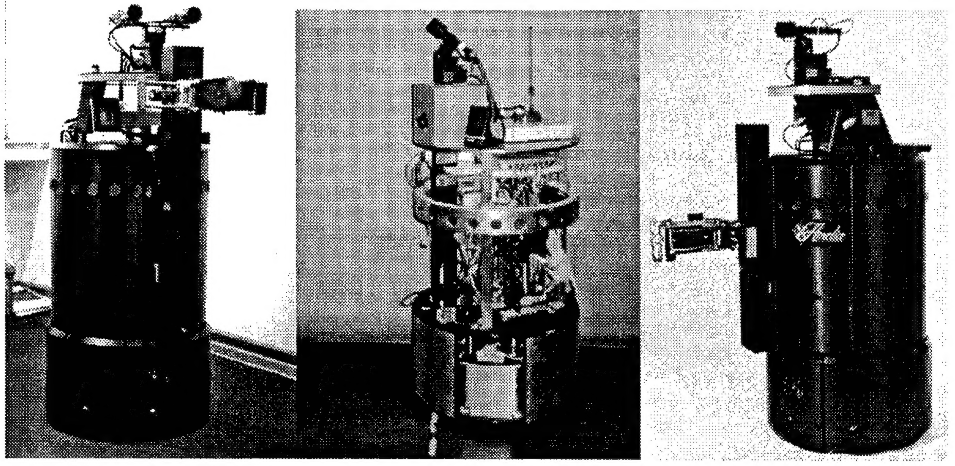


Figure 1: The robots used in our research: RHINO (University of Bonn), XAVIER, and AMELIA (both CMU).

to building topological maps, described in Section 3. Subsequently, Section 4 evaluates the utility of the integrated approach empirically. The paper is concluded by a discussion in Section 5.

2 Grid-Based Maps

The metric maps considered here are discrete, two-dimensional occupancy grids, as originally proposed in [6, 19] and since implemented successfully in various systems. Each grid-cell $\langle x, y \rangle$ in a map has a value attached that measures the subjective belief that this cell is occupied. More specifically, it contains the belief whether or not the center of the robot can be moved to the center of that cell (it represents the *configuration space* of the robot, see *e.g.*, [16]). Occupancy values are determined based on sensor readings.

This section describes the four major components of our approach to building grid-based maps [31]:

1. **Interpretation.** Sensor readings are mapped to occupancy values.
2. **Integration.** Multiple sensor interpretations are integrated over time, to yield a combined estimate of occupancy.

3. **Position estimation.** The position of the robot is constantly monitored and errors are corrected.

4. **Exploration.** Shortest path through unoccupied regions are generated to move the robot towards unexplored terrain.

Examples of metric maps are shown in various places in this paper.

2.1 Sensor Interpretation

To build metric maps, sensor reading must be "translated" into occupancy values $occ_{x,y}$ for each grid cell $\langle x, y\rangle$. The idea here is to train an artificial neural network [25] using Back-Propagation to map sonar measurements to occupancy values [31]. As shown in Figure 2, the input to the network consists of

- two values that encode $\langle x, y\rangle$ in polar coordinates relative to the robot (angle to the first of the four sensors, and distance), and
- the four sensor readings closest to $\langle x, y\rangle$.

The output target for the network is 1, if $\langle x, y\rangle$ is occupied, and 0 otherwise. Training examples can be obtained by operating a robot in a known environment, and recording its sensor readings; notice that each sonar scan can be used to construct many training examples for different x-y coordinates. In our implementation, training examples are generated by a mobile robot simulator.

Once trained, the network generates values in [0, 1] that can be interpreted as probability[1] for occupancy. Figure 3 shows three examples of sonar scans (top row, bird's eye view) along with their neural network interpretation (bottom row). The darker a value in the circular region around the robot, the larger the occupancy value computed by the network. Figure 3a&b show situations in a corridor. Here the network predicts the walls correctly. Notice the interpretation of the erroneous long reading in the left side of Figure 3a, and the erroneous short reading in 3b. For the area covered by those readings, the network outputs roughly 0.5, which indicates its uncertainty. Figure 3c shows a different situation in which the interpretation of the sensor values is less straightforward. This example illustrates that the network interprets sensors in the context of neighboring sensors. Long readings are only interpreted as free-space, if the neighboring sensors agree. Otherwise, the network returns values close to 0.5, which again indicates uncertainty. Situations such as the

[1]It has been shown that, under certain assumption, a neural network trained to predict a binary random variable approaches the probability distribution of this random variable [11, 36]

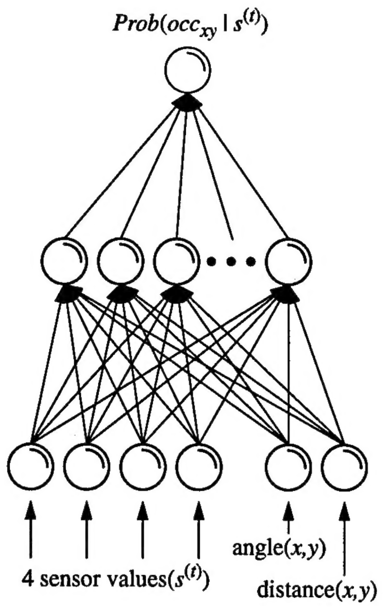


Figure 2: An artificial neural network maps sensor measurements to probabilities of occupancy.

one shown in Figure 3c—that defy simple interpretation—are typical for cluttered indoor environments.

Training a neural network to interpret sonar sensors has two key advantages over hand-crafted approaches to sensor interpretation:

1. Since neural networks are trained based on examples, they can easily be adapted to new circumstances. For example, the walls in the competition ring of the 1994 AAAI robot competition [29] were much smoother than the walls in the building in which the software was originally developed. Even though time was short, the neural network could quickly be retrained to accommodate this new situation. Others, such as Pomerleau [23], also report a significant decrease in development time of integrated robotic systems through the use of machine learning algorithms.

2. Multiple sensor reading are interpreted simultaneously. Most current approaches interpret each sensor reading individually, one-by-one, which amounts to making a conditional independence assumption between adjacent sonar

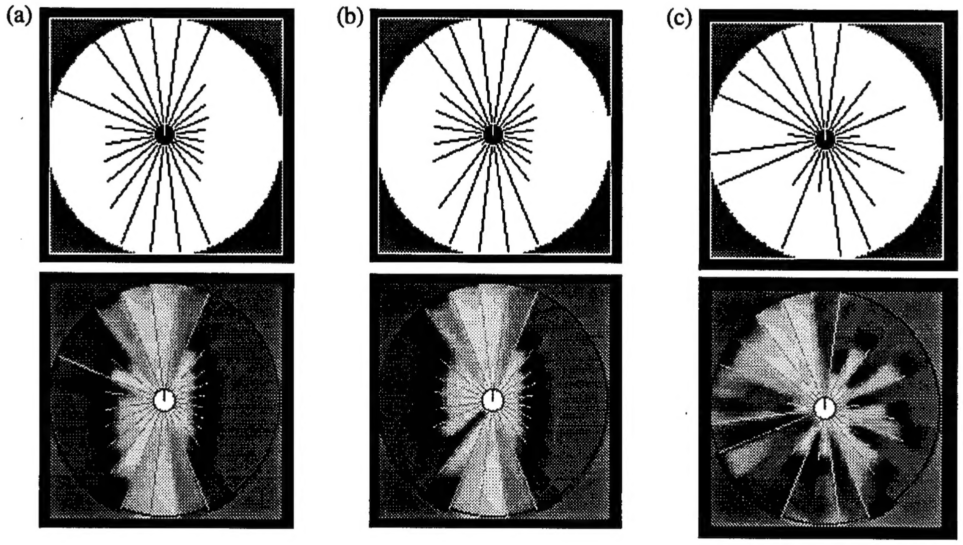


Figure 3: Sensor interpretation: Three example sonar scans (top row) and local occupancy maps (bottom row), generated by the neural network. Bright regions indicate free-space, and dark regions indicate walls and obstacles (enlarged by a robot diameter).

sensors. This assumption is questionable in practice. For example, the reflection properties of most surfaces depend strongly on the angle of the surface to the sonar beam, which can only be detected by interpreting multiple sonar sensors simultaneously.

2.2 Integration Over Time

Sonar interpretations must be integrated over time, to yield a single, consistent map. To do so, it is convenient to interpret the network's output for the t-th sensor reading (denoted by $s^{(t)}$) as the *probability* that a grid cell $\langle x, y \rangle$ is occupied, conditioned on the sensor reading $s^{(t)}$:

$$Prob(occ_{x,y}|s^{(t)})$$

A map is obtained by integrating these probabilities for all available sensor readings, denoted by $s^{(1)}, s^{(2)}, \ldots, s^{(T)}$. In other words, the desired occupancy value for each grid call$\langle x, y\rangle$ can be written as the probability

$$Prob(occ_{x,y}|s^{(1)}, s^{(2)}, \ldots, s^{(T)}),$$

which is conditioned on *all* sensor reading. A straightforward approach to estimating this quantity is to apply Bayes' rule. To do so, one has to assume independence of the noise in different readings. More specifically, given the true occupancy of a grid cell $\langle x, y\rangle$, the conditional probability $Prob(s^{(t)}|occ_{x,y})$ must be assumed to be independent of $Prob(s^{(t')}|occ_{x,y})$ if $t \neq t'$. This assumption is not implausible—in fact, it is commonly made in approaches to building occupancy grids. The desired probability can now be computed as follows:

$$Prob(occ_{x,y}|s^{(1)}, s^{(2)}, \ldots, s^{(T)}) =$$
$$1 - \left(1 + \frac{Prob(x)}{1-Prob(x)} \prod_{\tau=1}^{T} \frac{Prob(occ_{x,y}|s^{(\tau)})}{1-Prob(occ_{x,y}|s^{(\tau)})} \frac{1-Prob(x)}{Prob(x)}\right)^{-1}$$

Here $Prob(x)$ denotes the prior probability for occupancy (which, if set to 0.5, can be omitted in this equation). The derivation of this formula is straightforward and can be found in [19, 21]. Notice that this formula can be used to update occupancy values incrementally. An example map of a competition ring constructed at the 1994 AAAI autonomous robot competition is shown in Figure 4.

2.3 Position Estimation

The accuracy of the metric map depends crucially on the alignment of the robot with its map. Unfortunately, slippage and drift can have devastating effects on the estimation of the robot position. Identifying and correcting for slippage and drift is therefore an important issue in map building [9, 24].

Figure 5 gives an example that illustrates the importance of position estimation in grid-based robot mapping. In Figure 5a, the position is determined solely based on dead-reckoning. After approximately 15 minutes of robot operation, the position error is approximately 11.5 meters. Obviously, the resulting map is too erroneous to be of practical use. Figure 5b is the result of exploiting and integrating three sources of information:

1. **Wheel encoders.** Wheel encoders measure the revolution of the robot's wheels. Based on their measurements, odometry yields an estimate of the robot's position at any point in time. We will denote this estimate by $\langle x^*_{\text{robot}}, y^*_{\text{robot}}, \theta^*_{\text{robot}}\rangle$. As can be seen from Figure 5a, odometry is very accurate over short time intervals, but inaccurate in the long run.

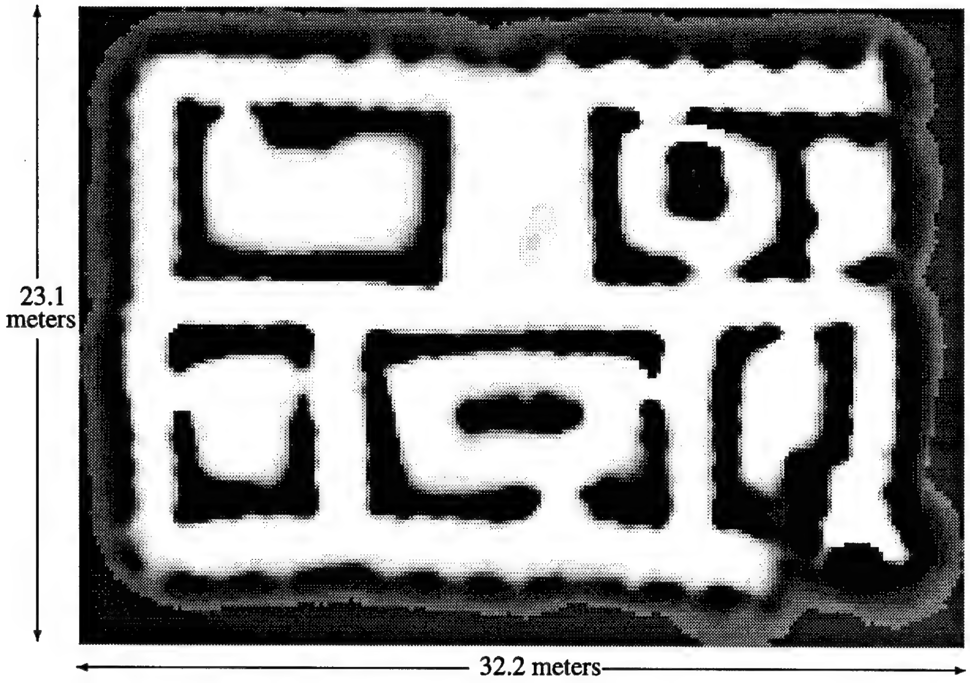
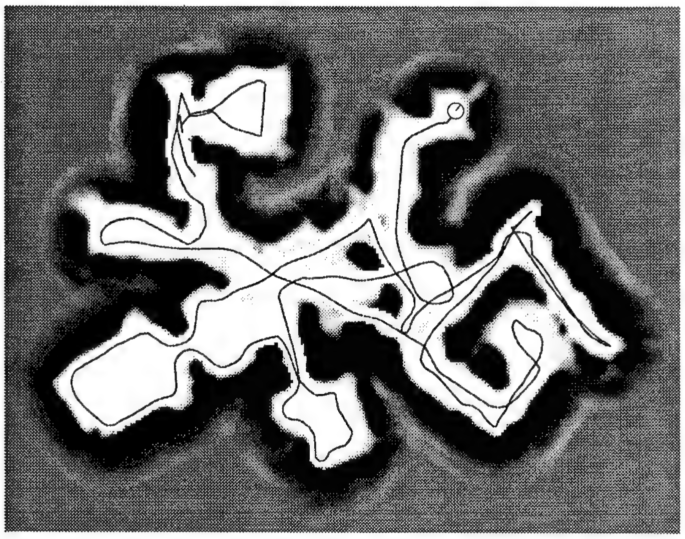


Figure 4: Grid-based map, constructed at the 1994 AAAI autonomous mobile robot competition with the techniques described here.

2. **Map correlation.** Whenever the robot interprets an actual sensor reading, it constructs a "local" map (such as the ones shown in Figure 3). The *correlation* of the local and the corresponding section of the global map is a measure of their correspondence [27]. Obviously, the more correlated both maps are, the more similar the local map looks to the global one, hence the more plausible it is. The correlation is a function of the robot's position $\langle x^*_{\text{robot}}, y^*_{\text{robot}}, \theta^*_{\text{robot}}\rangle$. Thus, the correlation of the local with the global map gives a second source of information for aligning the robot's position.

3. **Wall orientation.** A third component memorizes the *global wall orientation* [4, 12]. This approach rests on the restrictive assumption that walls are either parallel or orthogonal to each other, or differ by more than 15 degrees from these canonical wall directions. In the beginning of map building, the global orientation of walls is estimated by analyzing consecutive sonar scans (*cf.* Figure 6). This is done by searching straight lines that connect the

(a)


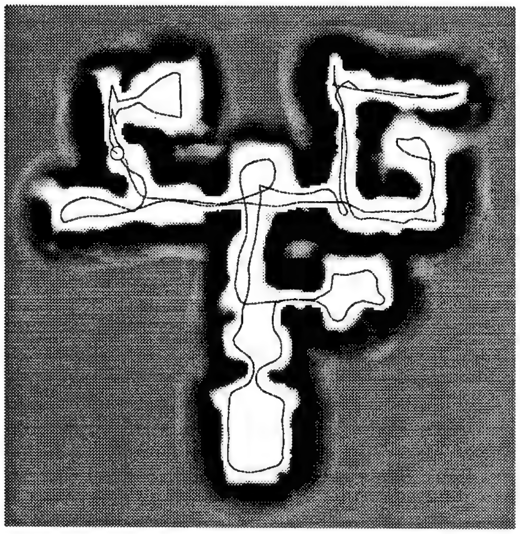
(b)


Figure 5: Map constructed without (a) and with (b) the position estimation mechanism described in this paper. In (a), only the wheel encoders are used to determine the robot's position. The positional error accumulates to more than 11 meters, and the resulting map is clearly unusable. This illustrates the importance of sensor-based position estimation for map building.

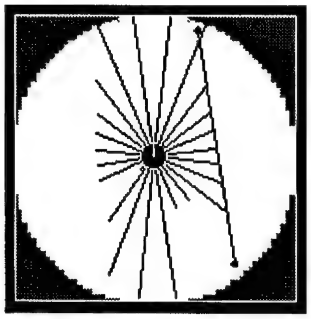


Figure 6: Wall, detected by considering five adjacent sonar measurements. Wall orientations are used to correct for dead-reckoning errors in the robot orientation θ_{robot}.

endpoints of five or more adjacent sonar measurements. Once the global wall orientation (denoted by θ_{wall}) has been determined, subsequent sonar are used to realign the robot's orientation. More specifically, suppose the robot detects a line in a sonar scan. Let $\hat{\theta}$ be the angle of this line relative to the robot. Then—in the ideal case—

$$\alpha \;:=\; (\theta_{\text{robot}} + \hat{\theta} - \theta_{\text{wall}}) \;\textit{modulo}\; 90^\circ$$

should be zero, *i.e.*, , the detected wall should be orthogonal or parallel to θ_{wall}. If this is not the case, the robot corrects its orientation accordingly,*i.e.*, by minimizing

$$\sigma(\alpha) \;:=\; \begin{cases} (|\alpha| - 15^\circ)^2 & \text{if } |\alpha| \leq 15^\circ \\ 0 & \text{if } |\alpha| > 15^\circ \end{cases}$$

This function is graphically depicted in Figure 7. As can be seen by the shape of the curve, small deviations are weighted the most, and wall orientations are only incorporated when they do not deviate from the expected orientation by more than 15°. We found empirically that this position estimation mechanism is extremely robust to noise and errors in wall detection.

All three mechanisms basically provide a probability density for the robot's position [33]. The exact function that is being minimized when calculating the robot's position is the following:

$$J \;:=\; \beta_1 \left[(x^*_{\text{robot}} - x_{\text{robot}})^2 + (y^*_{\text{robot}} - y_{\text{robot}})^2 \right]$$

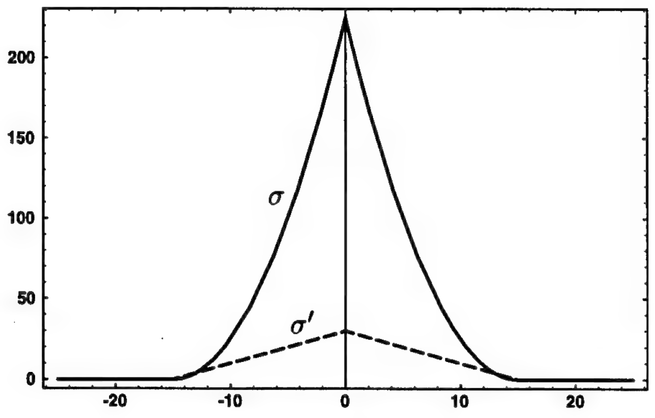


Figure 7: The function σ and its derivative. σ is most sensitive to values close to zero. Thus, small deviations of the expected and observed wall orientation have the strongest effect. If this deviation is larger than 15°, it is completely ignored. Consequently, walls that deviate from the expected wall orientation by more than 15° have no effect.

$$\begin{aligned} &+ \quad \beta_2 \left(\theta^*_{\text{robot}} - \theta_{\text{robot}}\right)^2 \\ &- \quad \beta_3 \; corr(x_{\text{robot}}, y_{\text{robot}}, \theta_{\text{robot}}) \\ &- \quad \beta_4 \; \sigma(\theta_{\text{robot}} + \hat{\theta} - \theta_{\text{wall}}) \end{aligned} \tag{1}$$

Here β_1, β_2, β_3, and β_4 are gain parameters that trades off the different sources of information. The first two terms in (1) correspond to the odometry of the robot. The third term measures the correlation between the global and the local map, and the fourth term relates the global wall orientation to the observed wall orientation. In our implementation, Equation (1) is differentiable, and gradient descent is employed to minimize J. When a new sonar reading arrives, the previous gradient search is terminated and its result is incorporated into the current position estimation.

Position control based on odometry and map correlation alone (items 1 and 2 above) works well if the robot travels through mapped terrain, but seizes to function if the robot explores and maps unknown terrain. The third mechanism, which arguably relies on a restrictive assumption concerning the nature of indoor environments, has proven extremely valuable when autonomously exploring and mapping large-scale indoor environments. Notice that all maps shown in this paper (with the exception of the map shown in Figure 5a) have been generated using this position estimation mechanisms.

2.4 Exploration

To autonomously acquire maps, the robot has to explore. The idea for (greedy) exploration is to let the robot always move on a minimum-cost path to the nearest unexplored grid cell; The cost for traversing a grid cell is determined by its occupancy value. The minimum-cost path is computed using a modified version of *value iteration*, a popular dynamic programming algorithm [1, 13]:

1. **Initialization.** Unexplored grid cells are initialized with 0, explored ones with ∞:

   $$V_{x,y} \longleftarrow \begin{cases} 0, & \text{if } \langle x,y \rangle \text{ unexplored} \\ \infty, & \text{if } \langle x,y \rangle \text{ explored} \end{cases}$$

   Grid cells are considered *explored* if their occupancy value $Prob(occ_{x,y})$ has been updated at least once. Otherwise, they are *unexplored.*

2. **Update loop.** For all explored grid cells $\langle x,y \rangle$ do:

   $$V_{x,y} \longleftarrow \min_{\substack{\xi=-1,0,1\\ \zeta=-1,0,1}} \{V_{x+\xi,y+\zeta} + Prob(occ_{x+\xi,y+\zeta})\}$$

   Value iteration updates the value of all explored grid cells by the value of their best neighbors, plus the costs of moving to this neighbor (just like A* [20]). Cost is here equivalent to the probability $Prob(occ_{x,y})$ that a grid cell $\langle x,y \rangle$ is occupied. The update rule is iterated. When the update converges, each value $V_{x,y}$ measures the *cumulative cost* for moving to the nearest unexplored cell. However, control can be generated at any time [5], long before value iteration converges.

3. **Determine motion direction.** To determine where to explore next, the robot generates a minimum-cost path to the unexplored. This is done by steepest descent in V, starting at the actual robot position. Determining the motion direction is done in regular time intervals, and is fully interleaved with updating V.

Figure 8a shows V after convergence, using the map shown in Figure 8b. All white regions are unexplored, and the grey-level indicates the cumulative costs V for moving towards the nearest unexplored point. Notice that the all minima of the

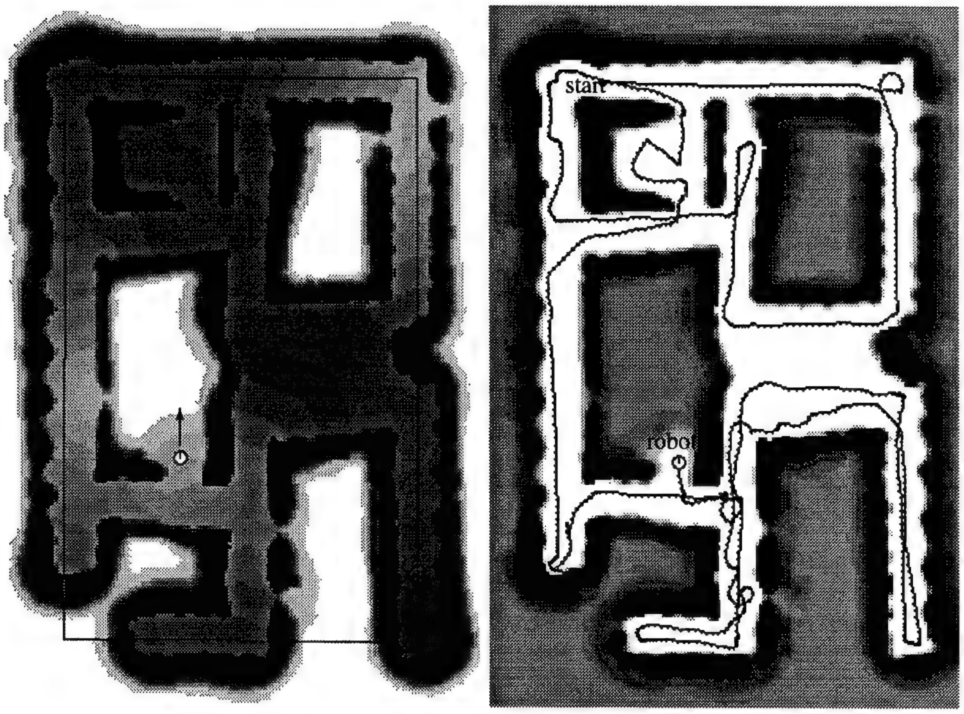


Figure 8: **Autonomous exploration.** (a) Exploration values V, computed by value iteration. White regions are completely unexplored. By following the grey-scale gradient, the robot moves to the next unexplored area on a minimum-cost path. (b) Actual path traveled during autonomous exploration, along with the resulting metric map. The large black rectangle in (a) indicates the global wall orientation θ_{wall}.

value function correspond to unexplored regions—there are no local minima. For every point $\langle x, y\rangle$, steepest descent in V leads to the nearest unexplored area.

Unfortunately, plain value iteration is too inefficient to allow the robot to explore in real-time. Strictly speaking, the basic value iteration algorithm can only be applied if the cost function does not increase (which frequently happens when the map is updated). This is because if the cost function increases, previously adjusted values V might become too small. While value iteration quickly decreases values that are too large, *increasing* values can be arbitrarily slow [31]. Consequently, the basic value iteration algorithm requires that the value function be initialized completely (Step 1) whenever the map—and thus the cost function—is updated. This is very inefficient, since the map is updated almost constantly. To avoid complete re-initializations, and to further increase the efficiency of the approach, the basic paradigm was extended in the following way:

4. **Selective reset phase.** Every time the map is updated, only values $V_{x,y}$ that are too small are identified and reset. This is achieved by the following loop, which is iterated:

   For all explored $\langle x, y \rangle$ do:

$$V_{x,y} \longleftarrow \infty \qquad \text{if} \qquad V_{x,y} < \min_{\substack{\xi=-1,0,1 \\ \zeta=-1,0,1}} \left\{ \begin{array}{l} V_{x+\xi,y+\zeta} \ + \\ Prob(occ_{x+\xi,y+\zeta}) \end{array} \right\}$$

   Notice that the remaining $V_{x,y}$-values are not affected. Resetting the value table in this way bears close resemblance to the value iteration algorithm described above.

5. **Bounding box.** To focus value iteration, a rectangular bounding box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is maintained that contains all grid cells in which $V_{x,y}$ may change. This box is easily maintained in the value iteration update. As a result, value iteration focuses on small fraction of the grid only, hence converges much faster. Notice that the bounding box bears similarity to *prioritized sweeping* [18].

Figure 8b shows a snapshot of autonomous exploration in the environment depicted in Figure 4. The right plot, 8b, sketches the path taken during autonomous exploration. At the current point, the robot has already explored the major hallways, and is about to continue exploration of a room. Circular motion, such as found in the bottom of this plot, occur when two unexplored regions are about equally far away (=same costs). Notice that the complete exploration run shown here took less than 15 minutes. The robot moved constantly, and frequently reached a velocity of 80 to 90 cm/sec (see also [3, 10]).

Value iteration is a very general procedure, which has several properties that make it attractive for real-time mobile robot navigation:

- **Any-time algorithm.** As mentioned above, value iteration can be understood as an any-time planner [5]. Any-time algorithms are able to make decisions regardless of the time spent for computation. The more time that is available, however, the better the results. Value iteration allows the robot to explore in real-time.

- **Full exception handling.** Value iteration pre-plans for arbitrary robot locations. This is because V is computed for every location in the map, not just the current location of the robot. Consequently, the robot can quickly react if it finds itself to be in an unexpected location, and generate appropriate motion directions without any additional computational effort. This is particularly important in our approach, since the robot uses a fast routine for avoiding collisions with obstacles, which may modify the motion direction commanded by the planner at its own whim [10].

- **Multi-agent exploration.** Since value iteration generates values for all grid-cells, it can easily be used for collaborative multi-agent exploration.

- **Point-to-point navigation.** By changing the initialization of V (Step 1), the same approach is used for point-to-point navigation [31].

In grid maps of size 30 by 30 meters, optimized value iteration, done from scratch, requires approximately 2 to 10 seconds on a SUN Sparc station. In cases where the selective reset step does not reset large fractions of the map (which is the common situation), value iteration converges in less than a second. For example, the planning time in the map shown in Fig. 4 typically under 2 seconds, usually under a tenth of a second. In the light of these results, one might be inclined to think that grid-based maps are sufficient for autonomous robot navigation. However, value iteration (and similar planning approaches) requires time quadratic in the number of grid cells, imposing intrinsic scaling limitations that prohibit efficient planning in large-scale domains. Due to their compactness, topological maps scale much better to large environments. In what follows we will describe our approach for deriving topological graphs from grid maps.

3 Topological Maps

3.1 Constructing Topological Maps

Topological maps are built on top of the grid-based maps. The key idea is simple but very effective: The free-space of a grid-based map is partitioned into a small number of regions, separated by *critical lines*. Critical lines correspond to narrow passages such as doorways. The partitioned map is then mapped into an isomorphic graph. The precise algorithm is illustrated in Figure 9, and works as follows:

1. **Thresholding.** Initially, each occupancy value in the occupancy grid is thresholded. Cells whose occupancy value is below the threshold are con-

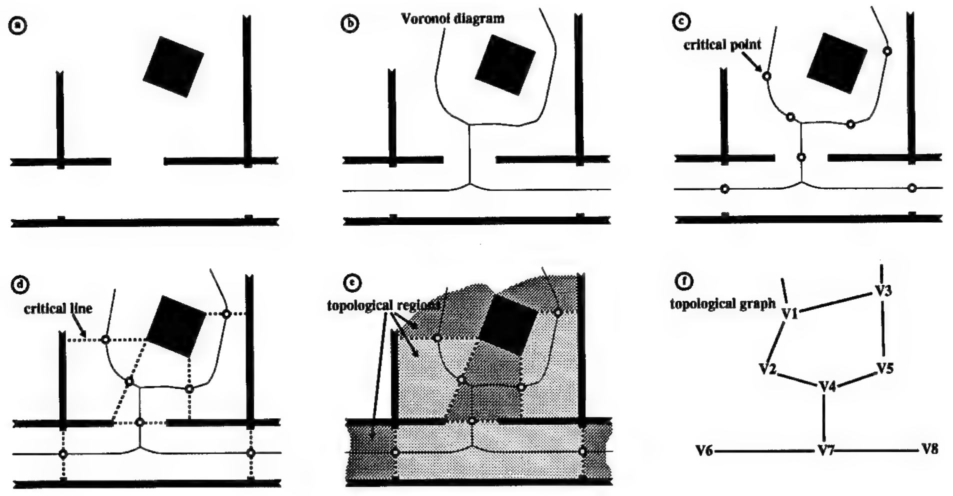


Figure 9: **Extracting topological maps.** (a) Metric map, (b) Voronoi diagram, (c) critical points, (d) critical lines, (e) topological regions, and (f) the topological graph.

sidered free-space (denoted by C). All other points are considered occupied (denoted by $\bar{C}$).

2. **Voronoi diagram.** For each point in free-space $\langle x, y \rangle \in C$, there is one or more *nearest point(s)* in the occupied space $\bar{C}$. We will call these points the *basis points of* $\langle x, y \rangle$, and the distance between $\langle x, y \rangle$ and its basis points the *clearance of* $\langle x, y \rangle$. The Voronoi diagram [16] is the set of points in free-space that have at least two different (equidistant) basis-points. Figure 9b depicts a Voronoi diagram.

3. **Critical points.** The key idea for partitioning the free-space is to find *"critical points."* Critical points $\langle x, y \rangle$ are points on the Voronoi diagram that minimize clearance locally. In other words, each critical point $\langle x, y \rangle$ has the following two properties: (a) it is part of the Voronoi diagram, and (b) the clearance of all points in an ε-neighborhood of $\langle x, y \rangle$ is *not* smaller. Figure 9c illustrates critical points.

4. **Critical lines.** Critical lines are obtained by connecting each critical point with its basis points (*cf.* Figure 9d). Critical points have exactly two basis points (otherwise they would not be local minima of the clearance function).

Critical lines partition the free-space into disjoint regions (see also Figure 9e).

5. **Topological graph.** The partitioning is mapped into an isomorphic graph. Each region corresponds to a node in the topological graph, and each critical line to an arc. Figure 9f shows an example of a topological graph.

Critical lines are motivated by two observations. Firstly, when passing through a critical line, the robot is forced to move in a considerably small region. Hence, the loss in performance inferred by planning using the topological map (as opposed to the grid-based map) is considerably small. Secondly, narrow regions are more likely blocked by obstacles (such as doors, which can be open or closed).

Figure 10 illustrates the topological map extracted from the grid-based map depicted in Figure 4. Figure 10a shows the Voronoi diagram of the thresholded map, and Figure 10b depicts the critical lines (the critical points are on the intersections of critical lines and the Voronoi diagram). The resulting partitioning and the topological graph are shown in Figure 10c&d. As can be seen, the free-space has been partitioned into 67 regions. Additional examples of metric and topological maps are shown in Figures 11 and 12. These maps are partitioned into 22 (Figure 11c&d) and 39 regions (Figure 12c&d).

3.2 Planning with Topological Maps

The enormous compactness of topological maps—when compared to the underlying grid-based map—facilitates efficient planning. To replace the grid-based planner by a topological planner, the planning problem is split into three subproblems, all of which can be tackled separately and very efficiently.

1. **Topological planning.** First, paths are planned using the abstract, topological map. Shortest paths in the topological maps can easily been found using one of the standard graph search algorithms, such as Dijkstra's or Floyd/Warshal's shortest path algorithm, A*, or dynamic programming. In our implementation, we used the value iteration approach described in Section 2.4.

2. **Triplet planning.** To translate topological plans into motion commands, a so-called "triplet planner" generates (metric) paths for each set of three adjacent topological regions in the topological plan. More specifically, let $T_1, T_2, \ldots, T_n$ denote the plan generated by the topological planner, where each T_i corresponds to a region in the map. Then, for each triplet

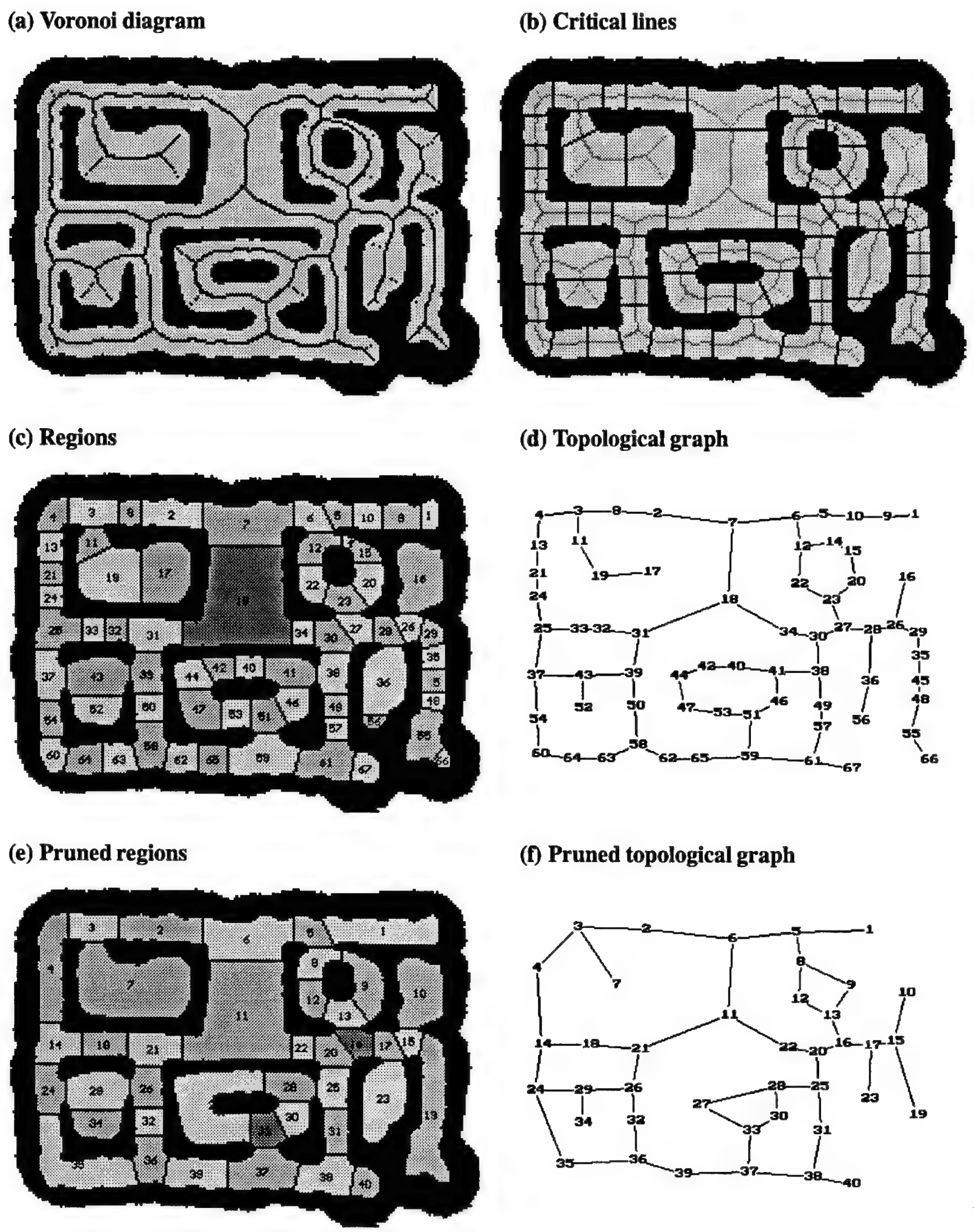
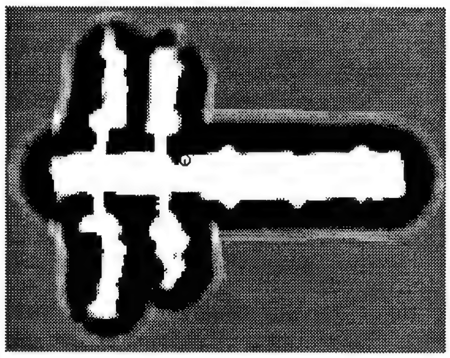
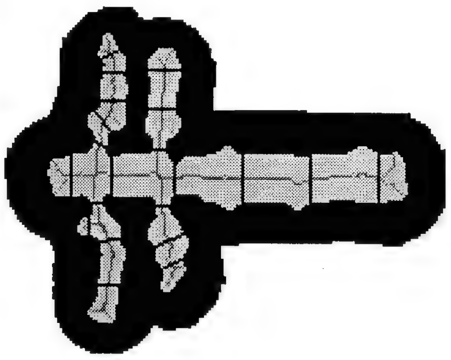


Figure 10: Extracting the topological graph from the map depicted in Figure 4: (a) Voronoi diagram, (b) Critical points and lines, (c) regions, and (d) the final graph. (e) and (f) show a pruned version (see text).

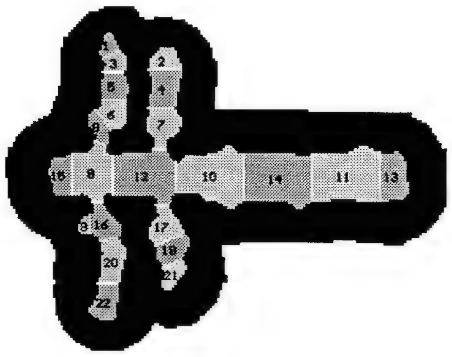
(a) Grid-based map



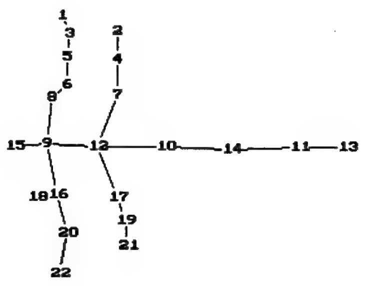
(b) Voronoi diagram and critical lines



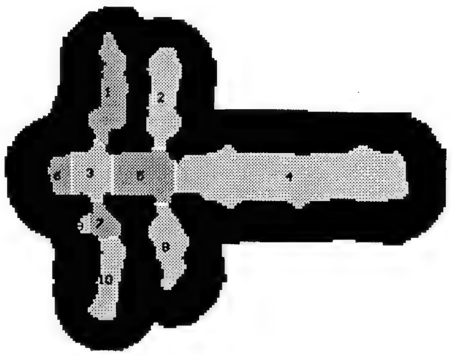
(c) Regions



(d) Topological graph



(e) Pruned regions



(f) Pruned topological graph

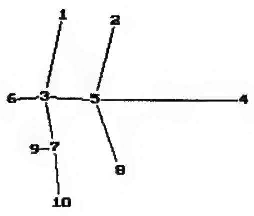
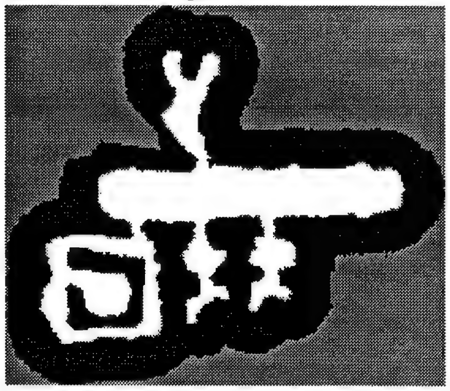
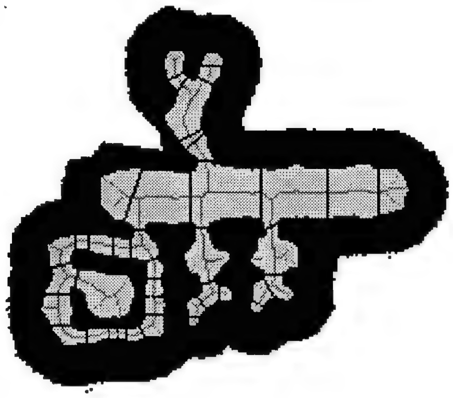


Figure 11: Another example of an integrated grid-based, topological map.

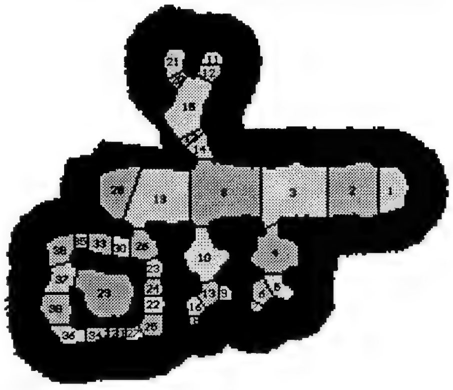
(a) Grid-based map



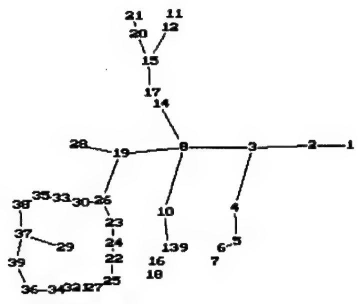
(b) Voronoi diagram and critical lines



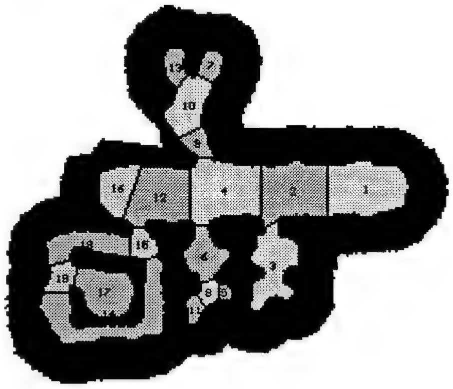
(c) Regions



(d) Topological graph



(e) Pruned regions



(f) Pruned topological graph

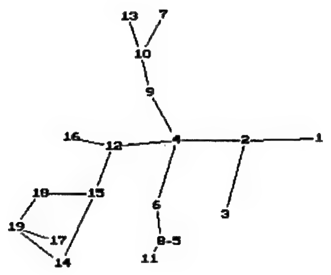


Figure 12: A third example.

$\langle T_i, T_{i+1}, T_{i+2} \rangle$ ($i = 1, \ldots, n-1$ and $T_{n+1} := T_n$), and each grid cell in T_i, the triplet planner generates shortest paths to the cost-nearest point in T_{i+2} in the grid-based map, under the constraint that the robot exclusively moves through T_i and T_{i+1}. For each triplet, all shortest paths can be generated in a single value iteration run: Each point in T_{i+2} is marked as a (potential) goal point (just like the unexplored points in Section 2.4), and value iteration is used to propagate costs through T_{i+1} to T_i just as described in Section 2.4. Triplet plans are used to "translate" the topological plan into concrete motion commands: When the robot is in T_i, it moves according to the triplet plan obtained for $\langle T_i, T_{i+1}, T_{i+2} \rangle$. When the robot crosses the boundary of two topological regions, the next triplet plan $\langle T_{i+1}, T_{i+2}, T_{i+3} \rangle$ is activated.

The triplet planner can be used to move the robot to the region that contains the goal location.

3. **Final goal planning.** The final step involves moving to the actual goal location, which again is done with value iteration. Notice that the computational cost for this final planning step does not depend on the size of the map. Instead, it depends on the size and the shape of the final topological region T_n, and the location of the goal.

The key advantage of this decomposition is that almost all computation can be done off-line, for all path planning problems. For example, the map shown in Figure 10, which is the most complex map investigated here, has 67 topological nodes. Thus, there are only $67 \times 66 = 4422$ topological plans, only half of which must be memorized (since topological plans are symmetric). The map also has approximately 200 triplets, for which all triplet plans are easily computed. Thus, by decomposing the planning in a topological planning problem and a triplet planning problem, and by pre-computing and memorizing all topological and triplet plans, path planning amounts to table-lookup.

However, it should be noted that the topological decomposition does not change the (worst-case) complexity of the planning problem, so that all one can hope for is a constant speed-up. Assuming that the number of topological regions grows linearly with the size of the grid-based map, and assuming that the size of each region does not depend on the size of the map, topological planning using value iteration is quadratic in the size of the environment (just like planning using the grid-based map). The computational complexity of computing all triplet plans is linear in the length of the path and hence in the size of the map), as is the computation of all final goal-plans. In fact, computing *all* triplet plans and *all* final goal plans is still linear in the size of the grid-based map, so that the topological planner is the only non-linear component in the approach proposed here. Although both—regular grid-based planning and topological planning—require in the worst case time quadratic in the size of the world, the fact that topological maps are orders of magnitude more compact leads to a relative difference of several orders of magnitude. This huge difference is important in practice.

4 Performance Results

Topological maps are abstract representations of metric maps. As is generally the case for abstract representations and abstract problem solving, there are three criteria for assessing the appropriateness of the abstraction: *consistency*, *loss*, and *efficiency*.

1. **Consistency.** Two maps are consistent with each other if every solution (plan) in one of the maps can be represented as a solution in the other map.

2. **Loss.** The loss measures the loss in performance (path length), if paths are planned in the more abstract, topological map as opposed to the grid-based map.

3. **Efficiency.** The efficiency measures the relative time complexity of problem solving (planning).

Typically, when using abstract models, efficiency is traded off with consistency and performance loss.

4.1 Consistency

The topological map is always consistent with the grid-based map. For every abstract plan generated using the topological map, there exists a corresponding plan in the grid-based map (in other words, the abstraction has the *downward solution property* [26]). Conversely, every path that can be found in the grid-based map has an abstract representation which is a admissible plan in the topological map (*upward solution property*). Notice that although consistency appears to be a trivial property of the topological maps, not every topological approach proposed in the literature generates maps that are consistent with their corresponding metric representation.

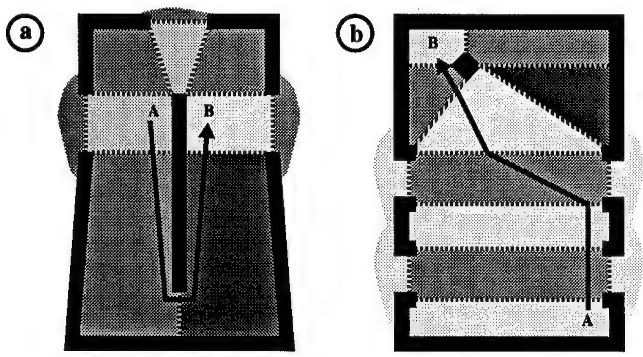


Figure 13: Two examples in which the approach presented here yields suboptimal results. In both cases, the problem is to plan a path from "A" to "B". (a) The topological planner will chose a sub-optimal path, since it leads only through two intermediate regions (as opposed to three). Such situations occur only if the topological graph contains cycles (which correspond to isolates obstacles in the thresholded grid-based map). (b) The triplet planner fails to move the robot on a straight line, since it looks only two topological regions ahead.

4.2 Loss

Abstract representations lack detail. Thus, paths generated from topological maps may not be as short as paths found using the metric representation. For example, Figure 13a shows a situation in which a topological planner would chose a detour, basically because of the different sizes and shapes of the topological regions. Figure 13b depicts a situation in which the triplet-planner would give non-optimal results, since is determines the motion direction based on a limited look-ahead.

To measure the average performance loss, we empirically compared shortest paths found in a metric map with those generated using the corresponding topological approach, for each of the three maps shown in Figures 4, 10, 11, and 12. The results are summarized in Table 2. For example, for the map shown in Figures 4 and 10d, we conducted a total of 23,881,062 experiments, each using a different starting and goal position that were generated systematically with an evenly-spaced grid. The results are intriguing. Planning with the topological map increases the length of the paths by an average of 3.24%. In other words, the average length of a shortest path is 15.88 meters, which increases on average by 0.51 meters if robot motion is planned using the topological map. 0.28 meters (1.82%) are due to sub-

	map 1 (Figs. 4,10)	map 2 (Figures 11)	map 3 (Figures 12)
grid cells	27,280	20,535	19,236
resolution	15 cm	10 cm	15 cm
cycles	8	0	1
topological regions	67	22	39
triplets	626	184	352
average shortest path length			
... meters	15.87	9.42	11.55
... grid-cells	94.2	84.8	68.5
... topological regions	7.84	4.82	6.99
average loss			
... due to topological planning	1.82%	0.00%	0.03%
... due to triplet-planning	1.42%	1.19%	1.28%
... **total loss**	**3.24%**	**1.19%**	**1.31%**
total experiments	23,881,062	1,928,540	4,576,435
complexity			
... grid-based planning	$2.56 \cdot 10^6$	$1.74 \cdot 10^6$	$1.32 \cdot 10^6$
... topological planning	525	106	273
... **difference (factor)**	$\mathbf{4.89 \cdot 10^3}$	$\mathbf{1.64 \cdot 10^4}$	$\mathbf{4.83 \cdot 10^3}$

Table 2: Survey of the results.

optimal choices by the topological planner, and the remaining 0.23 meters (1.42%) are due to suboptimal action choices made by the triplet planner. It is remarkable that in 83.4% of all experiments, the topological planner returns a loss-free plan. The largest loss that we found in our experiments was 11.98 meters, which was observed in 6 of the 23,881,062 experiments.

Figure 14a shows the average loss as a function of the length of the shortest path. As can be seen there, for shorter paths the loss is a monotonically increasing function of the path length. As the path length exceeds 22.5 meters, the loss decreases. We attribute the latter observation to the fact that these paths are among the longest possible paths given the size of the environment, thus even a topological planner cannot increase the length of these paths any further.

The empirical loss for the maps shown in Figure 11 and 12 is even smaller, partially because there are fewer cycles in those maps. As summarized in Table 2, the average loss for the map depicted in Figure 11 is 1.19%, and the average loss for the map shown in Figure 12 is 1.31%. Figures 15a and 16a depict the loss as a function of optimal path length. Notice because there are no cycles in the second map (Figure 11), the topological planner always produces the optimal plan (*i.e.*, a plan that includes the shortest path). Consequently, the 1.19% loss

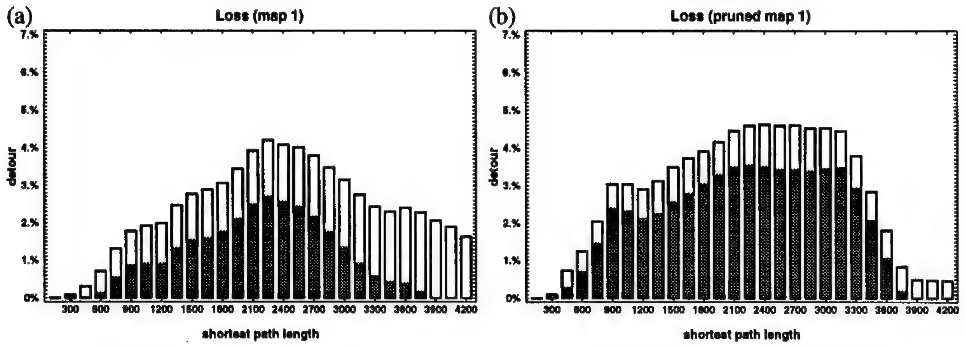


Figure 14: Loss for paths generated for the map shown in Figures 4 and 10, using (a) the regular and (b) the pruned topological map. They grey portion of the loss is due to suboptimal action choices by the topological planner, while the white portion is due to the triplet representation.

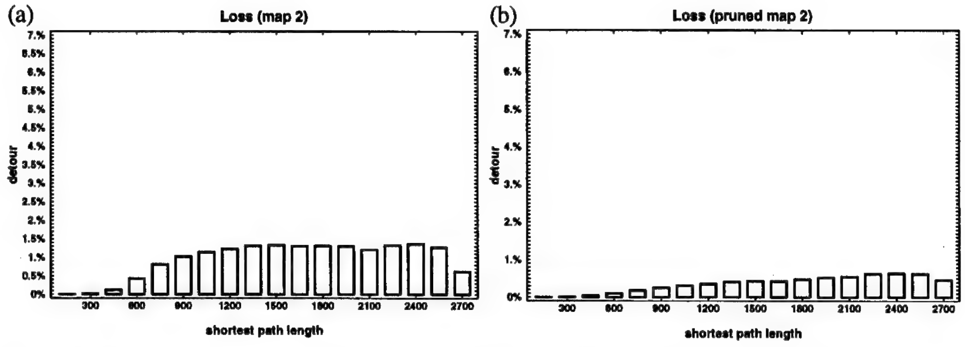


Figure 15: Paths generated for the map shown in Figure 11, using (a) the regular and (b) the pruned topological map.

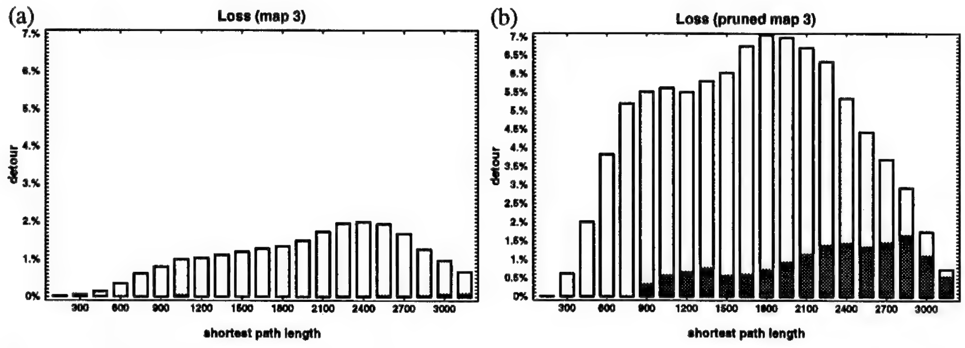


Figure 16: Paths generated for the map shown in Figure 12, using (a) the regular and (b) the pruned topological map.

	map 1 (Figs. 4,10)	map 2 (Figures 11)	map 3 (Figures 12)
grid cells	27,280	20,535	19,236
resolution	15 cm	10 cm	15 cm
cycles	8	0	1
topological regions	40	10	19
triplets	222	30	166
average shortest path length			
... meters	15.87	9.42	11.55
... grid-cells	94.2	84.8	68.5
... topological regions	6.12	3.25	4.65
average loss			
... due to topological planning	3.11%	0.00%	0.83%
... due to triplet-planning	0.94%	0.37%	5.22%
... **total loss**	**4.05%**	**0.37%**	**6.05%**
total experiments	23,881,062	1,928,540	4,576,435
complexity			
... grid-based planning	$2.56 \cdot 10^6$	$1.74 \cdot 10^6$	$1.32 \cdot 10^6$
... topological planning	245	32.5	88.4
... **difference (factor)**	$\mathbf{1.05 \cdot 10^4}$	$\mathbf{5.36 \cdot 10^4}$	$\mathbf{1.49 \cdot 10^4}$

Table 3: Results for the pruned maps.

can be exclusively attributed to suboptimal action choices by the triplet planner. The 1.31% loss for the map shown in Figure 12 is mostly due to the triplet planner (1.23%), although rare topological detours infer an additional loss of 0.03%. Graphs illustrating the relative loss as a function of shortest path length are shown in Figures 15a and 16a.

We also investigated even more compact representations, such as those shown in Figures 10e&f, 11e&f, and 12e&f. These maps were obtained by *pruning* the original topological map: Pairs of adjacent regions are combined into a single region, if neither of them has more than two neighbors. Pruning subsumes series of nodes in long corridors into a single node (such as nodes 4, 13, 21, and 24 in Figure 10c&d), and also eliminates certain end-nodes (such as the region 17, 56, and 66, in 10c&d). The results of experiments measuring the loss for these pruned maps are summarized in Table 3. For example, in 23,881,062 experiments using the pruned graph depicted in Figure 10e&f, the average loss was 0.64 meters (4.05%), which is 26.1% larger than the loss inferred by the un-pruned graph. For the map shown in Figures 11e&f, pruning actually *reduced* the overall loss to 0.37% (0.03 meters), which is only 31.0% of the loss inferred by the un-pruned map. Finally, the pruned map shown in Figure 12e&f produces an average detour of 5.18% (0.70

meter), which is significantly larger (361%) than the loss inferred by the un-pruned map—this difference is due to the fact that a long corridor is pruned into a single topological entity in Figure 12e&f. Figures 14b, 15b, and 16b depict the loss for the pruned map as a function of optimal path length. The shape of the curves here are similar to those obtained for the un-pruned maps. Figure 15 illustrates once again that pruning reduces the loss for the cycle-free second map. We conclude that the pruned graph is generally more compact, but unless the metric map is free of cycles, pruning increases the average loss.

4.3 Efficiency

The most important advantage of topological planning lies in its efficiency. Value iteration is quadratic in the number of grid cells. For example, the map shown in Figure 4 happens to possess 27,280 explored cells. In the average case, the number of iterations of value iteration is roughly equivalent to the length of the shortest path, which in our example map is 94.2 cells. Thus, in this example map, value iteration requires on average $2.57 \cdot 10^6$ backups. Planning using the topological representation is several orders of magnitudes more efficient. The average topological path length is 7.84. Since the topological graph shown in Figure 10d has 67 nodes, topological planning requires on average 525 backups. Notice the enormous gain in efficiency! Planning using the metric map is $4.89 \cdot 10^3$ more expensive than planning with the topological map. In other words, planning on the topological level increases the efficiency by more than three orders of magnitude, while inducing a performance loss of only 3.24%.

The computational reduction is even more dramatic for the pruned maps, such as the one shown in Figure 10e&f. This map consists of 40 nodes, and the average topological path length is 6.12. Consequently, topological planning is $1.05 \cdot 10^4$ more efficient than planning with the metric map, which is more than twice as efficient as planning with the un-pruned map. However, as can be seen by comparing the results shown in Table 2 and 3, the performance loss induced by the pruned map is 25% larger than the loss inferred by the un-pruned map.

The map shown in Figure 11, which is smaller than the other maps but was recoded with a higher resolution, consists of 20,535 explored grid cells and 22 topological regions (un-pruned map), or 10 regions (pruned map). On average, paths in the grid-based map lead through 84.8 cells. The average length of a topological plan is 4.82 (un-pruned map), or 3.25 (pruned map, averaged over 1,928,540 systematically generated path planning problems). Here the complexity reduction is even more significant than in the first example. Planning using the metric map is a factor of $1.64 \cdot 10^4$ more expensive than planning with the topological map when using the un-pruned map. This factor increases to $5.36 \cdot 10^4$ when using the pruned map. Clearly, since the pruned map exhibits a smaller loss, it is superior to the un-pruned version in both categories: loss and efficiency.

Similar results are obtained for the map depicted in Figure 12. Here the planning complexity is reduced by a factor of $4.83 \cdot 10^3$ (un-pruned map), or $1.49 \cdot 10^4$ (pruned map). While these numbers are empirical and only correct for the particular maps investigated here, we conjecture that the relative quotient is roughly correct for other maps as well.

It should be noted that in our implementation, every topological plan is pre-computed and memorized in a look-up table. Our most complex example maps contain 67 nodes, hence there are only 2,211 different plans that are easily generated and memorized. If a new path planning problem arrives, topological planning amounts to looking up the correct plan.

5 Discussion

This paper proposes an integrated approach to mapping indoor robot environments. It combines the two major existing paradigms: grid-based and topological. Grid-based maps are learned using artificial neural networks and Bayes' rule. Topological maps are generated by partitioning the grid-based map into critical regions.

Building occupancy maps is a fairly standard procedure, which has proven to yield robust maps at various research sites. To the best of our knowledge, the maps exhibited in this paper are significantly larger than maps constructed from sonar sensors by other researchers. Since neural networks interpret sonar readings in the context of adjacent sensor measurements, they does not make a commonly made conditional independence assumption between adjacent sensor measurements—resulting in more accurate interpretations of sonar measurements. This paper also demonstrates that by integrating multiple sources of information, the robot position can be tracked accurately and in real-time in environments of moderate size—which is crucial for building metric maps.

The most important aspect of this research, however, is the way topological graphs are constructed. Previous approaches have constructed topological maps from scratch, memorizing only partial metric information along the way. This often led to problems of disambiguation (*e.g.*, different places that look alike), and problems of establishing correspondence (*e.g.*, different views of the same place). This paper advocates to integrate both, grid-based and topological maps. As a direct consequence, different places are naturally disambiguated, and nearby locations are detected as such. In the integrated approach, landmarks play only an indirect role, through the grid-based position estimation mechanisms. Integration of landmark information over multiple measurements at multiple locations is automatically done in a consistent way. Visual landmarks, which often come to bear in topological approaches, can certainly be incorporated into the current approach, to further improve the accuracy of position estimation (see *e.g.*, [14, 32]). In fact, sonar sensors can be understood as landmark detectors that indirectly—through the grid-based map—help determine the actual position in the topological map (*cf.* [30]).

One of the key empirical results of this research concerns the cost-benefit analysis of topological representations. While grid-based maps yield more accurate control, planning with more abstract topological maps is several orders of magnitude more efficient. A large series of experiments showed that in a map of moderate size, the efficiency of planning can be increased by three to four orders of magnitude, while the loss in performance is negligible (*e.g.*, 1.82%). We believe that the topological maps described here will enable us to control an autonomous robot in multiple floors in our university building—complex mission planning in environments of that size was completely intractable with our previous methods.

Despite these encouraging results, there is a variety of important open questions that warrant future research.

- **Sensor dynamics.** The current approach does not account for sensor drift or sensor failure. Once trained, the weights of the interpretation network are frozen. However, in principle it is possible to use a map as to generate targets for the interpretation network. As a result, the robot could constantly re-adjust its own interpretations. Empirically, we have found our approach to be surprisingly robust with respect to the failure of sensors.

- **Other sensors.** A second goal of future research is to incorporate other types sensors. In an initial study, we extended the current approach by using a camera for floor segmentation, and 24 infrared light sensors that measure proximity by measuring the intensity of reflected light [3]. The Bayesian approach to sensor integration described in this paper is general flexible enough to accommodate other types of sensor information as well. In fact, in our initial experiments we found that the grid-based maps were more accurate when additional sensors were incorporated.

- **Dynamic environments.** Currently, we are unable to model environment dynamics (such as people, doors). It is an open question as to how to incorporate models of moving objects into a grid-base representation. A recent study [28] has demonstrated that "semi-dynamic obstacles" (these are obstacles such as doors, whose presence might change but which are tight to a certain location) can be modeled by a variance analysis of grid-cell values. Further research is warranted to evaluate the robustness and utility of such approaches.

A key disadvantage of grid-based methods, which is inherited by the approach presented here, is the need for accurately determining the robot's position. Since the difficulty of position control increases with the size of the environment, one might be inclined to think that grid-based approaches generally scale poorly to large-scale environments (unless they are provided with an accurate map). Although this argument is convincing, we are optimistic concerning the scaling properties of the approach taken here. The largest cycle-free map that was generated with this approach was approximately 100 meters long; the largest single cycle measured approximately 58 by 20 meters. We are not aware of any purely topological approach to robot mapping that would have been demonstrated to be capable of producing consistent maps of comparable size. Moreover, by using more accurate sensors (such as laser range finders), and by re-estimating robot positions backwards in time (which would be mathematically straightforward, but is currently not implemented because of its enormous computational complexity), we believe that maps can be learned and maintained for environments that are an order of magnitude larger than those investigated here.

The approach described here has become part of a larger software package that is now distributed through one of the major mobile robot suppliers in the US. It has successfully mapped many different environments. Part of the software package is a fast, reactive collision avoidance routine [10]. The advantage of integrating a fast collision avoidance routine is that dynamic obstacles and inaccuracies in the map do not lead to collisions. This module, combined with the mapping and planning approach described here, has found to navigate the robot reliably and with a speed of up to 90 cm/sec even in dynamic and cluttered environments (see [3]).

Acknowledgment

The authors wish to thank the RHINO mobile robot group at the University of Bonn, in particular Joachim Buhmann, Wolfram Burgard, Armin Cremers, Dieter Fox, Markus Giesenschlag, Thomas Hofmann, and Wolli Steiner for inspiring discussion and their contributions to the RHINO software system. Some of the low-level software (TCX [8] and device drivers) were provided by the XAVIER mobile robot group at Carnegie Mellon University, which is gratefully acknowledged. We thank Torsten Ihle for pointing out an error in a previous version of this paper, and we also acknowledge the steady and helpful support by Real World Interface Inc.

References

[1] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[2] J. Borenstein and Koren. Y. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991.

[3] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot Rhino. *AI Magazine*, 16(1), 1995.

[4] J. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 674–680, Scottsdale, AZ, May 1989.

[5] T. L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceeding of Seventh National Conference on Artificial Intelligence AAAI-92*, pages 49–54, Menlo Park, CA, 1988. AAAI, AAAI Press/The MIT Press.

[6] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[7] S. Engelson and D. McDermott. Error correction in mobile robot map learning. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2555–2560, Nice, France, May 1992.

[8] C. Fedor. TCX. An interprocess communication system for building robotic architectures. programmer's guide to version 10.xx. Carnegie Mellon University, Pittsburgh, PA 15213, December 1993.

[9] L. Feng, J. Borenstein, and H.R. Everett. "where am I?" sensors and methods for autonomous mobile robot positioning. Technical Report UM-MEAM-94-12, University of Michigan, Ann Arbor, MI, December 1994.

[10] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. Technical Report IAI-TR-95-13, University of Bonn, Institut für Informatik III, D-53117 Bonn, 1995.

[11] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley Pub. Co., Redwood City, California, 1991.

[12] R. Hinkel and T. Knieriemen. Environment perception with a laser radar in a fast moving robot. In *Proceedings of Symposium on Robot Control*, pages 68.1–68.7, Karlsruhe, Germany, October 1988.

[13] R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press and Wiley, 1960.

[14] D. Kortenkamp and T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 979–984, Menlo Park, July 1994. AAAI, AAAI Press/MIT Press.

[15] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Technical report, Department of Computer Science, University of Texas at Austin, Austin, Texas 78712, January 1990.

[16] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[17] M.J. Matarić. Interaction and intelligent behavior. Technical Report AI-TR-1495, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, 1994.

[18] A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[19] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[20] N. J. Nilsson. *Principles of Artificial Intelligence.* Springer Publisher, Berlin, New York, 1982.

[21] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[22] D. Pierce and B. Kuipers. Learning to explore and build maps. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1264–1271, Menlo Park, July 1994. AAAI, AAAI Press/MIT Press.

[23] D. A. Pomerleau. Knowledge-based training of artificial neural networks for autonomous robot driving. In J. H. Connell and S. Mahadevan, editors, *Robot Learning*, pages 19–43. Kluwer Academic Publishers, 1993.

[24] W.D. Rencken. Concurrent localisation and map building for mobile robots using ultrasonic sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2129–2197, Yokohama, Japan, July 1993.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II.* MIT Press, 1986.

[26] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[27] B. Schiele and J. Crowley. A comparison of position estimation techniques using occupancy grids. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1628–1634, San Diego, CA, May 1994.

[28] F. E. Schneider. Sensorinterpretation und Kartenerstellung für mobile Roboter. Master's thesis, Dept. of Computer Science III, University of Bonn, 53117 Bonn, December 1994. In German.

[29] R. Simmons. The 1994 AAAI robot competition and exhibition. *AI Magazine*, 16(1), Spring 1995.

[30] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of IJCAI-95*, Montreal, Canada, August 1995. IJCAI, Inc.

[31] S. Thrun. Exploration and model building in mobile robot domains. In E. Ruspini, editor, *Proceedings of the ICNN-93*, pages 175–180, San Francisco, CA, March 1993. IEEE Neural Network Council.

[32] S. Thrun. A bayesian approach to landmark discovery and active perception for mobile robot navigation. Technical Report CMU-CS-96-122, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213, April 1996.

[33] S. Thrun and A. Bücken. Learning maps for indoor mobile robot navigation. Technical Report CMU-CS-96-121, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213, April 1996.

[34] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, August 1996. AAAI, AAAI Press/MIT Press.

[35] M. C. Torrance. Natural communication with robots. Master's thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1994.

[36] V. Vapnik. *Estimations of dependences based on statistical data.* Springer Publisher, 1982.